Customized Infrastructures for Monitoring Business Processes

Marco Comuzzi
City University London
London, United Kingdom
Email: marco.comuzzi.1@city.ac.uk

Ruben Ivan Rafael Martinez
Eindhoven University of Technology
Eindhoven, The Netherlands
Email: r.i.rafael.martinez@student.tue.nl

***Abstract*—Process enactment technology provides native tools and add-ons for monitoring, such as APIs and monitoring consoles, which are usually highly entangled with the underlying process enactment logic and not customizable by process users. In such a case, all users access the same set of monitoring data and functions and process management resources may be allocated for monitoring concerns not of interest for users. In this context, we present a model and a tool for customized process monitoring infrastructures executing on top of existing process enactment technology. The model classifies the options about monitoring over which the preferences of process users may diverge. The tool implements the proposed model, generating customized process monitoring infrastructures embedding the business logic of the monitoring options chosen by process users.**

I. Introduction

Modern Workflow Management Systems (WfMS) provide native tools and add-ons for the monitoring of deployed business processes. WfMS log information about processes and executing process instances in a local database. The monitoring tools and add-ons are then built as a higher level interface of the database [1]. Monitoring data can vary from standard information, such as the status of instances and activities of active instances, to any process-specific variable logged by the WfMS, e.g. values of local and global process variables, or specific events, e.g. process variable updates or expiration of timeouts [2].

This approach to process monitoring presents limitations along the flexibility and scalability dimensions. About flexibility, it prevents monitoring customization, as all users get access to a common set of monitoring information tools that they may not need for their specific monitoring purposes. Different users, in fact, may have different monitoring requirements about the same process.

WfMS may also provide monitoring APIs that users can exploit for building their own monitoring infrastructure. Monitoring APIs are normally highly entangled with the underlying process technology [4], which makes them non reusable and hard to understand to a process developer unfamiliar with the underlying process technology. Moreover, APIs can support only the development of ad-hoc monitoring interfaces, which should evolve as the monitoring APIs evolve over time.

About scalability, the monitoring tools and APIs provided by WfMS clearly constitute a bottleneck of the monitoring activity, since all different monitoring requests have to be served by a single point of access to the data logged by the WfMS. In this regard, a non-customizable monitoring infrastructure is likely to lead to a waste of resources. By eliciting the users' customer requirements before the actual monitoring customization occurs, the business process provider can determine and instantiate only the resources strictly required for monitoring and create synergies among similar monitoring requests, e.g. reuse indexes for common queries on the process logging database. Such a dynamic use of resources is made also possible by modern cloud technology, which allows the provisioning of computing resources on-demand.

Note that increased flexibility of monitoring infrastructures is a pre-requisite for improved scalability. Before it would be possible to efficiently allocate resources to the monitoring concerns requiring it most, it is necessary to disentangle monitoring concerns from the underlying process technology. This will allow encapsulating monitoring concerns into process technology-agnostic services, which can be then deployed efficiently on an elastic computing platform.

This paper focuses on the *flexibility* issue, leaving scalability to be addressed by future work.

In particular, we devise a solution enabling the business process provider, i.e. the business entity running the WfMS, to provide customized monitoring tools to process users, satisfying their individual monitoring requirements. Monitoring requirements may vary for a variety of reasons, such as the need for monitoring established contracts [5], or predicting their violation [6], [7], or synchronizing internal and external processes of the customer organizations [4], [8]. In this paper we consider the cross-instance monitoring problem. In cross-instance monitoring, customer organizations (users) require monitoring across a set of instances, e.g. monitoring the average execution time of a given activity for all instances of a given process started in the past week. As such, cross-instance monitoring represents the basis of several management control activities, such as Business Activity Monitoring (BAM) [10].

The paper is structured as follows. We first provide a multi-dimensional model for cross-instance business process monitoring to structure the customization *solution space*, i.e. the space of offered monitoring options over which users' preferences may diverge (Section II). Then, we provide an implementation of the model in the form of a Web-based application that automatically synthesizes customized monitoring tools for the users (customer organizations) of the WfMS (Section III). Related work is discussed in Section IV, while we draw our conclusions and discuss future work in Section V.

II. A MODEL FOR CROSS-INSTANCE BUSINESS PROCESS MONITORING

The aim of our model is to capture the options for cross-instance process monitoring available to users in a multi-dimensional space. Such a space will serve as the blueprint for the implementation of the customized monitoring infrastructures specific to each user request. In Section II-A we discuss the rationale behind the design choices leading to our monitoring customization space, while Section II-B presents a formal characterization of our model.

A. Identifying monitoring dimensions for customization

A model for process monitoring customization aims at capturing the possible requirements of users of the process about monitoring, that is, understanding what could be monitored and how monitoring should occur. A first distinction within our model has thus to be made between the monitoring *object* and the monitoring *lifecycle*, which constitute the overarching dimensions of our monitoring model. The former identifies the monitoring information that users require from the process engine, whereas the latter identifies the modality according to which such information is acquired by or provided to the user over time.

For understanding the possible sub-dimensions and user options about the monitoring object, we rely on the integration of previous work in the area of Business Process Intelligence [11], BPMN extension for business activity monitoring [12], and on a model for business process analytics provided by the Workflow Management Coalition (WfMC) [1].

The work in [11] set the basis for identifying relevant process monitoring metrics for business intelligence. In the aim of building a data warehouse for business process monitoring, the paper identifies three types of monitoring data, i.e. Time, Status, and Resource data, which can be aggregated along the process granularity dimension. Granularity is defined at the level of process, service, and node. *Processes* invoke a set of *services* during their execution, and *services* are executed on physical *nodes*. The work in [12] extends BPMN for monitoring purposes identifying status and duration as main monitoring data and refining the granularity dimension considering also the process instance level and aggregation operators across the granularity levels, e.g. average, maximum, minimum, or frequency of occurrence (e.g. for status information, which is intrinsically not ordinal). The work in [1] mainly focuses on the status and time dimensions, providing a more detailed list of time-related variables relevant for monitoring and refining the list of possible statuses characterizing activities, instances, and processes.

To integrate the three proposal discussed above, in our monitoring model we consider *time* and *status* related monitoring variables. About granularity, in particular, we consider a simple process management model comprising *processes*, which are constituted by multiple *activities* and associated to multiple *instances* started by *users*. When an instance is executing, the activities belonging to the corresponding process are instantiated into *assigned* activities.

As far as status monitoring variables are concerned, our objective is to devise a generic process monitoring model, which can suit the case of both traditional workflow management systems and service-based business process enactment technology. Hence, we rely on the intersection between specifications proposed by the Workflow Management Coalition (WfMC) [1] and in the BPEL language [2], i.e. the de facto standard for service-based business processes.

Fig. 1. Instance (a) and Activity (b) state machines

Figure 1 shows finite state machines for instance and activity status that we consider in our monitoring model. Note that there are there are two final states for an instance: completed, which is reached when an instance is completed successfully, or failed, when the completion has not been successful, i.e. because of an error or because the user aborted the execution. Note that suspended instances can only terminate with a failure, whereas a running instance may either terminate successfully or fail. In particular, we excluded from our model the states defined for instance compensation in BPEL, as these do not find a counterpart in the WfMC specification and, consequently, in traditional workflow technology. Similarly, we excluded the *initiated* status defined by the WfMC as this is not defined in the BPEL state machine.

There is only one final state for activity execution, because the WfMC specification does not distinguish between states for successful and unsuccessful termination (as an unsuccessful termination should push the corresponding instance in the failed status).

As far as time-related monitoring variables are concerned, the time-related information in our monitoring model can be defined using status changes timestamps. As we will discuss later, in fact, process enactment technology usually logs the timestamps related to status changes of instances and activities. In particular, it has to be noted that, during their lifecycle, instances and activities may assume some states only once, whereas the state running and suspended can be assumed more than once.

For an instance ins_j we define the individual timestamps tt_j, tc_j, and tf_j representing the instant in time in which the instance has reached the status terminated, failed, or completed, respectively. Note that, depending on the status in which an instance terminates, tt_j coincides with either tc_j or tf_j. Similarly for an activity act_j we define the individual timestamps ti_j and tc_j representing the instants in time when an activity becomes inactive and completes, respectively.

About the statuses running and suspended, for both activities and instances, we can define a set of timestamps TR_j and TS_j representing the time instants in which an instance or activity becomes running or suspended, respectively:

[1] http://www.wfmc.org/reference-model.html

[2] http://pic.dhe.ibm.com/infocenter/dmndhelp/v8r0m1/index.jsp

TABLE I. TIME-RELATED MONITORING VARIABLES.

Granularity	Name	Defnition
instance	Duration	$DUR(ins_j) = tt_j - tr_{j,1}$
instance	Processing Time	$PIT(ins_j) = \sum_{i=0}^{I}(ts_{j,1} - tr_{j,i}) + \begin{cases} tt_j - tr_{j,I} & \text{if } \exists tr_{j,I}, \\ 0 & otherwise \end{cases}$
instance	Suspended Time	$SIT(ins_j) = \sum_{i=0}^{I-1}(tr_{j,i+1} - ts_j) + \begin{cases} tf_j - ts_{j,I} & \text{if } \exists tf_j, \\ 0 & otherwise \end{cases}$
activity	Duration	$DUR(act_j) = tc_j - tr_j$
activity	Waiting time	$WAT(act_j) = tr_{j,1} - ti_j$
activity	Turnaround time	$TUT(act_j) = tc_j - ti_j$
activity	Processing Time	$PAT(act_j) = \sum_{i=0}^{I}(ts_{j,i} - tr_{j,i}) + (tc_j - tr_j, I)$
activity	Suspended Time	$SAT(act_j) = \sum_{i=0}^{I-1}(tr_{j,i+1} - ts_{j,i}) + \begin{cases} tt_j - ts_{j,I} & \text{if } tt_j > tr_{j,I}, \\ tr_{j,I} - ts_{j,I} & otherwise \end{cases}$

$$TR_j = \{tr_{j,i}\}_{i=1,\ldots,I}, \text{ with } tr_{j,i+1} > tr_{j,i}, \forall i \quad (1)$$

$$TS_j = \{ts_{j,k}\}_{k=1,\ldots,K}, \text{ with } ts_{j,k+1} > ts_{k,j}, \forall j \quad (2)$$

Table I shows the time-related monitoring variables that we consider in our model. These are a restriction over the set of time-related process control variables defined in [1] on the basis of the states and related timestamps defined for instances and activities in our model.

For identifying the possible user options about the monitoring *lifecycle*, we refer to our previous work on customizable single-instance process monitoring [9]. In that work, we identified a standard lifecycle for business process monitoring, leading to the definition of two dimensions over which users can specify their options about monitoring, i.e Management and Notification

Management. Users may express options about the way monitoring data will be managed by the monitoring infrastructure before being provided. For instance, monitoring information can be stored in a batch or provided as soon as a new value is captured. The data may be stored, for instance to provide historical series to the user, or destroyed in the monitoring infrastructure once communicated to the user. Specifically, new values acquired can overwrite existing values (rewrite - *rw*) or being persisted (persist - *pe*) by the monitoring infrastructure. When notified to the user, monitoring values currently stored by the monitoring infrastructure may be consumed (*co*), i.e. they will be no longer available in the future, or they may be only read (*re*) and remain persisted by the monitoring infrastructure, e.g. to build historical time series. The options available to the user for the management monitoring design are derived from the combination of the modalities discussed before, that is, we identify a total of four possible options, i.e. *rw-co*, *rw-re*, *pe-co*, *pe-re*.

Notification. According to a well-recognized paradigm in distributed computing [13], there are two main ways monitoring data can be notified to the user. Users may pull information as they wish, or they may be pushed information by the monitoring infrastructure according to a pre-specified policy, e.g. periodically.

B. Formal model

The monitoring model discussed in the previous section can be formally characterized as presented in Table II. The formalization of Table II gives us a model to specify succinctly the monitoring customization options of users. Such a representation of users' monitoring requirements is exploited by the implementation of our tool (see Section III).

Specifically, a monitoring customization $c_{i,j}$ is a request to the process engine of the provider organization to monitor a certain (set of) variable(s) var over a possible restriction of the processes currently deployed within the engine. Such a restriction is defined on features characterizing the elements of the process management model introduced before, i.e. processes (pro), instances (ins), and activities (act).

For the pro, ins, and act sub-dimensions, we consider the following restriction options:

- ALL: it refers to all the processes/instances/activities currently associated to the user u_i by the process engine;
- [USER_SELECTION]: it restricts the set of processes/instances/activities to a list specified by the user, e.g. a set of process IDs;
- [TIME_FRAME]: it restricts the set of processes/instances/activities to the ones deployed/started/allocated within a specific time frame, i.e. before and/or after a specific date;

For the elements ins and act we consider also the following restriction options:

- [STATUS_BASED]: it restricts the set of instances/activities to the ones in a certain status at the moment a monitoring request will be issued to the process engine;

Note that the monitoring options are not mutually exclusive. For example, a user can specify options on both TIME_FRAME and STATUS_BASED to restrict monitoring of a certain variable to all instances (and/or activities) started in a certain timeframe and currently in a certain status, e.g. *running* or *terminated*.

TABLE II. MONITORING MODEL.

Element Name	Notation
Process Users	$U = u_1, \ldots, u_I$
Monitoring Customizations specified by users	$C = \{c_{i,j}\}_{\substack{i=1,\ldots,I \\ j=1,\ldots,J}}$ with $c_{i,j} = \langle MO, ML \rangle$
Monitoring Object (in a customization)	$MO = \langle var, pro, ins, act, agg \rangle$
Monitoring Variable	$var \in \{i\_status, a\_status, DIN, PIT, SIT, DAC, WAT, TUT, PAC, SAC\}$
Aggregation Options	$agg \in \{count, avg, max, min, sum\}$
Monitoring Lifecycle (in a customization)	$ML = \langle man, not \rangle$
Monitoring Management Options	$man \in \{rw - co, rw - re, pe - co, pe - re\}$
Monitoring Notification Options	$not \in \{push(policy), pull\}$

As previously discussed, the monitoring variables can relate to the status of instances, i_status, and activities, a_status, and to the time-related variables of Table I.

Eventually, the agg element in MO specifies the possible aggregation of monitoring data required over the restrictions applied. We adopt in this work the aggregation operators of the SQL language, i.e. $agg \in \{sum, avg, max, min, count\}$. Note that not all combinations of options are feasible. For instance, it is not possible to apply aggregation operators such as sum or avg to non-numeric values, e.g. activity or instance status. Similarly, operators max and min can be applied only to monitoring variables for which a total order relationship is defined, i.e. all the time related monitoring variables.

As far as the monitoring lifecycle ML is concerned, the Notification dimension not involves two possible possible options. The user can in fact $pull$ data from the monitoring infrastructure when needed, or the monitoring infrastructure can $push$ monitoring data to the user according to a certain policy. In this work we consider only periodic push of monitoring data. Hence, the $policy$ element represents the notification period, expressed in time units. For the Management dimension man we consider the four options discussed in the previous section.

As a sample, Eq. 3 captures the monitoring requirements of user u_1 interested in counting the number of activities in the instances of process p_1 started before January 1st and terminated in the status failed; u_1 wants to pull the information when required from the monitoring infrastructure and requires the monitoring information to be persisted by the monitoring infrastructure as long the customization is active (therefore, $man = \{pe - re\}$.

$$\begin{aligned} c_{1,1} &= \langle MO, ML \rangle \\ MO &= \{a\_status; p_1; \\ &\quad < 2013 - 01 - 01T00:00:00 + 00:00; \\ &\quad ALL; failed; count\} \\ ML &= \{pull; pe - re\} \end{aligned} \tag{3}$$

Eq. 4 captures the requirements of users u_2 interested in monitoring the average processing time of instances of all instances of the process p_2 terminated in the status completed. The user wants to be notified this information every six hours. Each time, the user requires only the average processing time of instances completed in the last six hours (that is, the user is not interested in building time series of monitoring data; $man = \{rw - co\}$).

$$\begin{aligned} c_{2,1} &= \langle MO, ML \rangle \\ MO &= \{PIT; p_2; ALL; -; completed; avg\} \\ ML &= \{push(6hr); rw - co\} \end{aligned} \tag{4}$$

III. A TOOL FOR CUSTOMIZED CROSS-INSTANCE PROCESS MONITORING

In this section we describe the tool we implemented. Our tool generates customized process monitoring infrastructures exploiting the monitoring model described in the previous section. We first provide a conceptual outline of our tool, in Section III-A, and then discuss its technical implementation in Section III-B.

A. Conceptual Architecture

The conceptual architecture of the tool is shown in Figure 2(a). The design aims at isolating components for each of the monitoring dimensions in the monitoring model of Table II. Conceptually, user interaction with the system can occur at customization time, i.e. when monitoring customizations are requested, and at run time, i.e. when the monitoring starts [14].

At customization time, users access the `Monitoring Customization Console (MCC)`. This comprises the `Customization Interface (CI)`, through which users specify their customization options. Customization requests are received by the `Monitor Generator (MG)`, which instantiates a `Custom Monitoring Infrastructure (CMI)` for each monitoring customization request. The `CMI` contains the `Monitoring Console (MC)`, through which users get access to the monitoring data according to the monitoring options specified in the customization. It also comprises three components in charge of realizing the business logic of the monitoring options identified in our model:

- `Query Engine (QE)`: It instructs the `BPE Handler` to capture the appropriate monitoring variable(s) requested by the user in the customization;

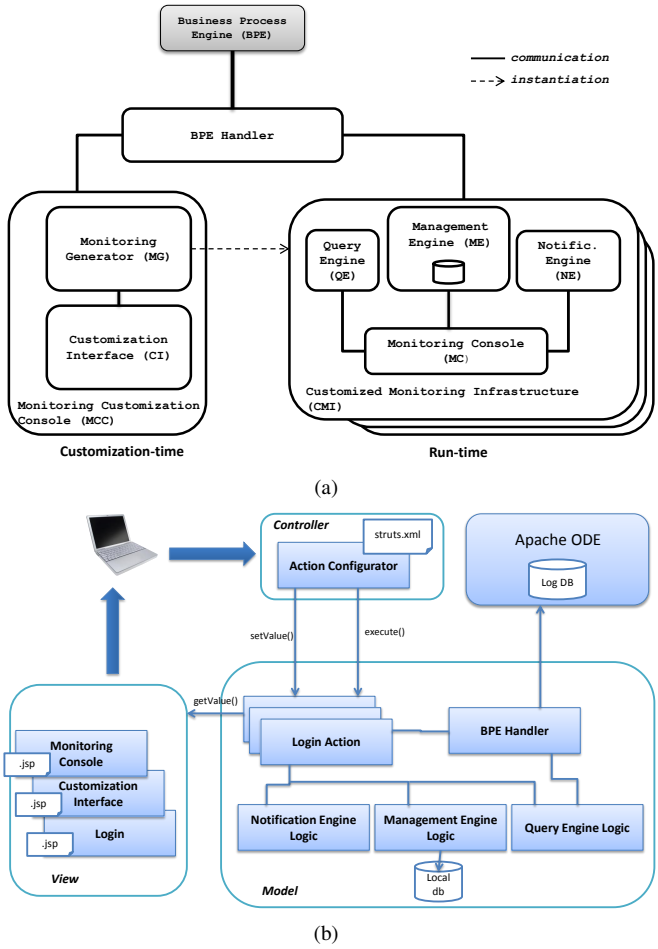

Fig. 2. Conceptual (a) and Technical (b) architecture of the implemented tool

- `Management Engine (ME)`: It realizes the logic chosen by the user about the management of the monitoring data obtained by the BPE; this module requires an internal database, since it may need to persist the data acquired from the BPE;
- `Notification Engine (NE)`: It realizes the logic chosen by the user about the notification of monitoring results (namely, push and pull).

Eventually, both customization time and run time modules interact with the BPE through the `BPE Handler`. This acts as the gateway between our tool and the specific business process technology chosen for the execution of monitored business processes.

At customization time, the `BPE Handler` allows the `MCC` to obtain information about running processes and related instances. At run time, it translates the instructions received by `QE` into executable queries over the logging database of the business process engine and sends back the query results obtained from the BPE.

The design pattern driving the design of the architecture of our tool is clearly the Model-View-Controller (MVC) pattern. `CI` and `MC` represent the *view* for customization- and run-time, respectively, as they are the interfaces through which the user can interact with our tool. The `MG` represents the *controller* in our architecture, connecting the interfaces to the `CMI`, which implements the *model* driving the cross-instance process monitoring, i.e. the monitoring options described in Section II.

B. Technical Realization

The tool has been implemented as a Java Web application, using Struts 2 as a framework supporting the MVC design pattern. We experimented with the Apache ODE BPEL engine, which is freely available and open source, that is, a specific `BPE Handler` has been developed for Apache ODE.

Figure 2(b) shows the technical architecture of our tool. The `MG` component in the conceptual architecture is implemented by the controller component. It comprises the `struts.xml` file, which is used to configure the appropriate actions in the model component to implement the monitoring options requested by the user. The controller implements the business logic of the customized monitoring infrastructure. Specifically, the BPE Handler issues query over the Apache ODE database, whereas the Management Engine component controls a local database where data acquired from Apache ODE may be stored in case the user requests monitoring data to be persisted.

The view part of our tool comprises a set of dynamic Web pages, which are configured by the data requested through the actions implemented in the controller. The tool comprises a page for the login of the user and pages implementing the Customization Interface and the Monitoring Console.

As introduced before, the main objective of the BPE Handler is to translate a monitoring customization expressed according to the formal model of Section II-B into a query for the Apache ODE database logging monitoring information. In particular, the translation process concerns the monitoring object MO element of a customization $c_{i,j}$.

A typical strategy adopted by current process management technology to log relevant process monitoring information is *event-based*. Relevant monitoring information in this case is captured in events logged by the BPE. Events refer to entities of the process management model, have a timestamp, and log different types of occurrences in the process engine, e.g. status changes, variable value changes, faults. An event-based logging strategy is followed by most process engines available on the market. As far as open source and freely available engines are concerned, this strategy is for instance adopted by the Apache ODE BPEL engine and YAWL. The event-based strategy is opposed to the *entity-based* strategy, adopted for instance by the OpenESB BPEL engine. The limitation of this strategy is that monitoring information is stored at the level of individual entities of the process management model, i.e. processes, instances, and activities. This prevents logging multiple status changes timestamps (in OpenESB, for instance, only the last status change is persisted at a given moment in time).

IV. Related Work

Data warehousing and OLAP (On Line Analytical Processing) [11] exploit the concept of cross-instance monitoring, but

taking a rather static perspective about monitoring, supporting the more static use case of executive decision making. In this work, we focus on a more dynamic perspective about monitoring, supporting scenarios such as cross-organizational business process synchronization [15] or online Business Activity Monitoring (BAM) [10].

The Monita [18] approach to process monitoring highlights the issue of entanglement between monitoring concerns and workflow specification and it proposes to implement the former as *aspects* blended within workflow specifications. The definition of monitoring aspects has to be performed by monitoring experts and cannot be delegated to process users, since the set of possible monitoring concerns is not abstracted into a high-level model understandable by non-expert users. A framework for ECA rules to support advanced process monitoring is presented in [19]. Also in this case, the expertise required for the specification of ECA monitoring rules prevents the delegation of the monitoring customization activity to process users.

Several approaches [2], [20], [21] to business process monitoring have been developed in the context of service-based business processes and, more specifically, processes specified using BPEL. An architecture for self-supervision of BPEL processes has been proposed in [2]. It involves the definition of *aspects* for the analysis and recovery of running BPEL processes, which are blended at runtime within the Active BPEL engine execution logic. This approach represents a typical example in which the resulting monitoring logic is highly entangled with the chosen process engine technology. The language defined for the definition of monitoring concerns, i.e. WsCol, is more expressive than our model of monitoring customization. Hence, it could be exploited to extend the range and complexity of user-defined monitoring variables and concerns of this work. A visual approach to specify monitoring queries on running BPEL processes is presented in [20]. Visual queries are then translated into one or more Xpath and SQL queries on the BPEL process specification and/or data logged by the BPEL engine. Although, also in this case, the range and complexity of monitoring concerns is higher when compared to our formal model, the approach is specific only to BPEL processes and does not specifically address the need for customizing the resulting monitoring infrastructure.

V. Conclusions and Outlook

This paper considers the issue of customized business process monitoring infrastructures in the context of cross-instance process monitoring. The design of our tool decouples the monitoring concern from the underlying process engine, making our approach generic, i.e. replicable using alternative process enactment technology. In this work we focused on the flexibility of current process monitoring technology. As stated in the Introduction, our future work will also concentrate on the scalability issue. In particular, we plan to work in two different directions. A first optimization will concern the access to the BPE database, such as indexes for highly requested monitoring configurations. A second optimization will concern the deployment of our Customized Monitoring Facilities on a cloud infrastructure. Individual CMIs can be deployed on virtualized appliances using only the resources that are strictly necessary, improving the performance of our tool.

References

[1] M. zur Muehlen and R. Shapiro, "Business process analytics," in *Handbook on Business Process Management 2*, 2010, pp. 137–157.

[2] L. Baresi and S. Guinea, "Self-supervising BPEL processes," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 247–262, 2011.

[3] M. zur Muehlen and D. T.-Y. Ho, "Risk management in the BPM lifecycle," in *Business Process Management Workshops*, 2006, pp. 454–466.

[4] W. Robinson and S. Purao, "Monitoring service systems from a language-action perspective," *IEEE Transactions on Services Computing*, vol. 4, pp. 17–30, 2011.

[5] M. Comuzzi and G. Spanoudakis, "Dynamic set-up of monitoring infrastructures for service based systems," in *ACM SAC*, 2010, pp. 2414–2421.

[6] D. Lorenzoli and G. Spanoudakis, "Predicting software service availability: Towards a runtime monitoring approach," in *IEEE ICWS*, 2011, pp. 736–737.

[7] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "Monitoring, prediction and prevention of SLA violations in composite services," in *Proc. 2010 IEEE International Conference on Web Services*, 2010, pp. 369–376.

[8] J. Simmonds, Y. Gan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse, "Runtime monitoring of Web service conversations," *IEEE Trans. Serv. Comput.*, vol. 2, pp. 223–244, July 2009.

[9] M. Comuzzi, S. Angelov, and J. Vonk, "Patterns to enable mass-customized business process monitoring," in *Proc. CAiSE*, no. 445-459, 2012.

[10] "Business activity monitoring: Calm before the storm," Gartenr Group Research, http://www.gartner.com/resources/105500/105562/105562.pdf, Tech. Rep. LE-15-9727, 2002.

[11] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.-C. Shan, "Business process intelligence," *Computers in Industry*, vol. 53, pp. 321–343, 2004.

[12] J.-P. Friedenstab, C. Janiesch, M. Matzner, and O. Muller, "Extending bpmn for business activity monitoring," in *Proc. 45th Hawaii ICSS*, 2012, pp. 4158–4167.

[13] L. Aldred, W. van der Aalst, M. Dumas, and A. ter Hofstede, "Dimensions of coupling in middleware," *Concurrency and Computation : Practice & Experience*, vol. 21, pp. 2233–2269, 2009.

[14] F. Gottschalk, W. van der Aalst, M. Jansen-Vullers, and M. La Rosa, "Configurable workflow models," *IJCIS*, vol. 17, pp. 177–221, 2008.

[15] R. Eshuis and P. Grefen, "Constructing customized process views," *Data and Knowledge Engineering*, vol. 64, no. 2, pp. 419–438, 2008.

[16] M. Reichert, S. Bassil, R. Bobrik, and T. Bauer, "The proviado access control model for business process monitoring components," *Enterprise Modelling and Information Systems Architectures*, vol. 5, no. 3, pp. 64–88, 2010.

[17] H. Reijers, R. Mans, and R. van der Toorn, "Improved model management with aggregated business process models," *Data Knowl. Eng.*, vol. 68, no. 1, pp. 221–243, 2009.

[18] O. Gonzalez, R. Casallas, and D. Deridder, "Monitoring and analysis concerns in workflow applications: from conceptual specifications to concrete implementations," *International Journal of Cooperative Information Systems*, vol. 20, no. 4, pp. 371–404, 2011.

[19] J. Bae, H. Bae, S.-H. Kang, and Y. Kim, "Automatic control of workflow processes using ECA rules," *IEEE TKDE*, vol. 16, no. 8, pp. 1010–1023, 2004.

[20] C. Beeri, A. Eyal, T. Milo, and A. Pilberg, "Query-based monitoring of BPEL business processes," in *Proceedings ACM SIGMOD*, 2007, pp. 1122–1124.

[21] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for WS-BPEL," in *Proc. World Wide Web Conference*, 2008.